Q53：最大子序和

1. 动态规划： f(i) : 以ai结尾的最大子序和。 f(i) = max(f(i-1)+ai, ai); 然后dp压缩，空间使用o(0)
2. 滑动窗口： 假设当前窗口和为sum, 最终结果为ans, 若当前sum <= 0 则如果对其继续扩张，sum对后边反而是负担，不会有贡献。所以舍弃该窗口，然后sum = ai, ans = max(ans,sum) 只有当sum > 0 时对sum扩张才有意义。

Q76：最小覆盖子串 s = "ADOBECODEBANC", t = "ABC" ans = "BANC" 即BANC是一个最小子串，包含了 A,B,C三个字符。时间要求o(n)

1. right指针遇到不属于t的字母就不断扩张，遇到属于t的字母就停下，然后left收缩
2. left收缩时，若指向属于t的字母，就判断去掉之后是否满足t的需要，满足就收缩，否则right继续扩张。ans在收缩后保留最小值。
3. 使用count记录频率到达t的字母的个数。当count到达t的字符频率就++， count和t的字符种类相同时，说明当前窗口满足t.
4. 窗口收缩扩展时，要动态的修改count的值。

窗口第二大的值

Q239：滑动窗口最大值 [1,3,-1,-3,5,3,6,7], k = 3 输出：[3,3,5,5,6,7]

1. 问题的关键是：若左端移出窗口，而他恰好是窗口内最大值。那么如何找到第二大的那个值。
2. 使用双端队列，的单调递减栈。若移出的元素不是队头那么直接删除即可，因为删除的不是窗口内最大值。
3. 若移出窗口元素是队头元素，说明删除元素是窗口内最大值。那么对头元素也要删除。然后以单调栈的方式把新元素加入队列尾部此时就可以确定新窗口的最大值，一定是队头。
4. 大的值入栈，会导致小的值出栈，恰好，小的值不需要。

Q283：移动零 把数组中的0移动到数组右侧，且不破坏非0元素的顺序。

1. left指向永远指向当前数组第一个0的位置。 right指向left后第一个非0元素，然后交换。
2. right再走向下一个非0元素，left走向下一个0.

Q395：至少有 K 个重复字符的最长子串 要求子串中每一个 字符的频率都大于等于k

1. **枚举字符**

```
1．对于字符x,若x的频率小于K,那么所有含有字符x的串都是非法的。
2．对当前串进行字符统计， 找到 频率小于 k的字符x。然后按照x字符进行分割。
3．若一个串找不到分割符，说明该串每一个字符频率都大于等于k．
4．对以上过程进行递归，每次都对当前串进行统计。
```

2. **枚举最长串的字符种类数量:滑动窗口**o(26n)

```
1．假设最长串，字符种类数量是1，那么我们根据字符种类数量控制窗口的大小。
2．right右移动，若字符种类数量没有超过p,那么right继续右移动。同时统计字符频率，还有频率超过k的字符。
3．若移动后字符种类大于 p,此时就要由 left右移动，使得种类数 =p为止。 同时更新频率，和频率超过k的字符。
4．tot 代表 [j, i] 区间所有的字符种类数量；sum 代表满足「出现次数不少于 k」的字符种类数量
5．如果 tot == sum ,说明该串合法了。即可取最大长度。
```

Q424 : 你可以将任意位置上的字符替换成另外的字符，总共可最多替换 k 次。在执行上述操作后，找到包含重复字母的最长子串的长度。

递增窗口 0. 我们枚举左边界。然后对其扩张。

1. 右边界单增，不回退。
2. 设当前窗口内频率最高的字符为 A，其频率为 maxn
3. right- left + 1 - maxn > k : 说明需要改变的字符数>k,无法通过替换K个字符使得其变为重复字符串。此时就要 left右移动一个。发现不满足时，当前left的最长子串就确定了，不能再增加了。所以left++，找以left++为左边界的最长子串。又因为二,以left++为左边界的，直接找更长的串即可。
4. right- left + 1 - maxn <= k : 说明以当前left左边界的串，还可以继续扩张。

```
int left = 0, right = 0;
while (right < n) {
    num[s.charAt(right) - 'A']++;
    // 打擂台求最大值 ， 是一个历史最大值。保存不是前一个窗口内的最大值。
    maxn = Math.max(maxn, num[s.charAt(right) - 'A']);
    if (right - left + 1 - maxn > k) {
        num[s.charAt(left) - 'A']--;
        left++;
    }
    right++;
}
return right - left;
```

5. 最后窗口的大小就是ans

Q1208 : 尽可能使字符串相等 将 s 中的第 i 个字符变到 t 中的第 i 个字符需要 |s[i] - t[i]| 的开销（开销可能为 0）

1. 也就是两个字符的 ASCII 码值的差的绝对值。用于变更字符串的最大预算是 maxCost，如果你可以将 s 的子字符串转化为它在 t 中对应的子字符串，则返回可以转化的最大长度。

```
while(right < len){
        // right指向的花费
        int rcost = Math.abs(s.charAt(right)-t.charAt(right));
        if(tcost >= rcost){
                tcost -= rcost; // 花费
                ++right;
        }else{  // tcost >= rcost 只要不够支付 右侧花费。就一直右移left并退费。
                // left指向的花费。
                int lcost = Math.abs(s.charAt(left)-t.charAt(left));
                tcost += lcost; // 退费
                ++left;
        }
        ans = Math.max(ans, right - left);
}
```